

# RIPASSO

---

Fissata una funzione  $g(n)$  possiamo dire:

$$O(g(n)) = \{ f(n) \mid \exists c \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 \quad f(n) \leq c * g(n) \}$$

Una funzione  $f(n)$  è detta *O grande di  $g(n)$*  se esiste una costante reale  $c$  tale che, per ogni punto successivo ad un certo punto  $n_0$ ,  $f(n)$  assume valori minori o uguali ai valori assunti da  $g(n)$  moltiplicata per  $c$ .

$$\Omega(g(n)) = \{ f(n) \mid \exists c \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 \quad c * g(n) \leq f(n) \}$$

Una funzione  $f(n)$  è detta *Omega grande di  $g(n)$*  se esiste una costante reale  $c$  tale che, per ogni punto successivo ad un certo punto  $n_0$ ,  $f(n)$  assume valori maggiori o uguali ai valori assunti da  $g(n)$  moltiplicata per  $c$ .

E ancora

$$\Theta(g(n)) = \{ f(n) \mid \exists c_1, c_2 \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 \quad c_1 * g(n) \leq f(n) \leq c_2 * g(n) \}$$

Una funzione  $f(n)$  è detta *Theta di  $g(n)$*  se esistono due costanti reali  $c_1$  e  $c_2$  tali che, per ogni punto successivo ad un certo punto  $n_0$ ,  $f(n)$  assume contemporaneamente

- valori maggiori o uguali ai valori assunti da  $g(n)$  moltiplicata per  $c_1$
- valori minori o uguali ai valori assunti da  $g(n)$  moltiplicata per  $c_2$

In parole povere, una funzione  $f(n)$  è detta *Theta di  $g(n)$*  se  $f(n)$  è contemporaneamente sia *Omega grande di  $g(n)$*  che *O grande di  $g(n)$* .

# PROGRAMMAZIONE DINAMICA

---

Abbiamo visto nel corso precedente come spesso cerchiamo la soluzione migliore di un algoritmo provando sia una soluzione iterativa che una ricorsiva, ma che facciamo quando nessuna di queste due da una soluzione accettabile?

L'idea su cui si basa la programmazione dinamica è evitare di calcolare più volte lo stesso risultato (come accade a volte negli algoritmi ricorsivi, ad esempio con Fibonacci) calcolandolo una sola volta per poi memorizzarlo per quando sarà necessario nuovamente.

# WEIGHTED INTERVAL SCHEDULING

---

Consideriamo un insieme di  $n$  attività

$$\{ 1, \dots, n \} \quad n \in \mathbb{N} \quad \forall \text{ attività } i \in \{ 1, \dots, n \} \text{ abbiamo } \begin{cases} s_i & \text{tempo inizio attività } i \\ f_i & \text{tempo fine attività } i \\ v_i & \text{valore/peso attività } i \end{cases}$$

Consideriamo due attività  $i$  e  $j$  compatibili se queste non si sovrappongono, ovvero se  $[s_i, f_i) \cap [s_j, f_j) = \emptyset$

Definiamo le seguenti funzioni:

1.  $C : \wp(\{1, \dots, n\}) \rightarrow \{ \text{true}, \text{false} \}$   
 $\forall A \subseteq \{1, \dots, n\} \quad C(A) = \begin{cases} \text{true} & \text{se } A \text{ contiene attività fra loro mutualmente compatibili} \\ \text{false} & \text{altrimenti} \end{cases}$
2.  $v : \wp(\{1, \dots, n\}) \rightarrow \mathbb{R}$   
 $\forall A \subseteq \{1, \dots, n\} \quad v(A) = \begin{cases} \sum_{i \in A} v_i & \text{se } A \neq \emptyset \\ 0 & \text{se } A = \emptyset \end{cases}$

Il problema WIS ci richiede di trovare, fra tutti i sottoinsiemi di attività mutualmente compatibili, quello con valore  $v$  massimo (è dunque un problema di ottimizzazione).

### ISTANZA

Un'istanza di tale problema è data dal numero di attività  $n$  e dalle triple associate ad ognuna delle  $n$  attività.

**SOLUZIONE:**  $S \subseteq \{1, \dots, n\}$  t.c.  $C(S) = \text{true} \wedge v(S) = \max\{v(A) \mid A \subseteq \{1, \dots, n\}\}$

Per risolvere un problema di programmazione dinamica dobbiamo innanzitutto individuare i sotto problemi più piccoli. Il generico sotto problema di WIS, identificato da  $i : 0 \leq i \leq n$  è nella forma:

**Dato  $X_i = \{1, \dots, i\}$  trovare  $S_i \subseteq X_i$  t.c.  $C(S_i) = \text{true} \wedge v(S_i) = \max\{v(A) \mid A \subseteq X_i\}$**

Assumiamo ora di aver già risolto tutti i sotto problemi più piccoli di  $i$  e cerchiamo di capire come calcolare  $S_i$  e  $v(S_i)$ .

Assumiamo anche che le attività siano ordinate per tempo di fine e definiamo

**$p(i) = \max\{j \mid j < i \wedge j \text{ è compatibile con } i\}$**

Esempio:

```

1  ____
2  _____
3      _____
4          _____
5      _____
6          _____
```

$v_i = 10, 2, 8, 1, 1, 3$

$p(i) = 0, 0, 1, 2, 1, 4$

Per calcolare  $S_i$  avremo quindi due casi:

1.  $i \in S_i$  allora  $S_i = S_{p(i)} \cup \{i\}$   
 $v(S_i) = v(S_{p(i)}) + v_i$
2.  $i \notin S_i$  allora  $S_i = S_{i-1}$   
 $v(S_i) = v(S_{i-1})$

Possiamo ora definire le equazioni di ricorrenza.

### CASO BASE

$i = 0$

$X_0 = \emptyset \implies S_0 = \emptyset$  e  $v(S_0) = 0$

## CASO PASSO

$i > 0$

$$v(S_i) = \max\{v(S_{p(i)}); v(S_{i-1})\}$$

$$S_i = \begin{cases} S_{p(i)} \cup \{i\} & \text{se } v(S_{p(i)}) + v_i \geq v(S_{i-1}) \\ S_{i-1} & \text{altrimenti} \end{cases}$$

Detto questo, è facile creare un algoritmo ricorsivo basato su queste equazioni, il problema di tale algoritmo però è che finirebbe per calcolare più volte lo stesso risultato portando il tempo ad essere esponenziale. La programmazione dinamica ci permette di memorizzare i risultati parziali dei sotto problemi più piccoli in modo da riutilizzarli senza doverli calcolarli nuovamente.

## DEFINIZIONE VARIABILI

Definiamo quindi  $M[i] \forall i$  t.c.  $0 \leq i \leq n$  come la soluzione del sotto problema  $i$ -esimo, ovvero come il massimo valore che posso ottenere prendendo un insieme di attività mutualmente incompatibili considerando solo le prime  $i$  attività.

## ALGORITMO

Scriviamo quindi l'algoritmo WIS definito come

```
WIS(){
    M[0] = 0
    for i = 0 to n
        M[i] = max(M[p[i]] + v[i], M[i-1])
    return M
}
```

Nell'esempio fatto prima l'algoritmo farebbe i seguenti passaggi

$$M[1] = \max(M[0] + 10, M[0])$$

$$M[2] = \max(M[0] + 2, M[1])$$

$$M[3] = \max(M[1] + 8, M[2])$$

$$M[4] = \max(M[2] + 1, M[3])$$

$$M[5] = \max(M[1] + 1, M[4])$$

$$M[6] = \max(M[4] + 3, M[5])$$

**M**

| 0 | 1  | 2  | 3  | 4  | 5  | 6  |
|---|----|----|----|----|----|----|
| 0 | 10 | 10 | 18 | 18 | 18 | 21 |

Possiamo poi stampare gli intervalli attraverso il seguente algoritmo

```
PrintW(i)
    if i != 0
        if v[i] + M[p[i]] > M[i - 1]
            print i
            PrintW(p[i])
        else
            PrintW(i - 1)
```

## LONGEST COMMON SUBSEQUENCE

---

Per alcuni esercizi è possibile trovare immediatamente una soluzione in programmazione dinamica, mentre per altri bisogna risolvere un problema associato che ci possa consentire di arrivare alla soluzione.

## SEQUENZE

Definiamo  $X = \langle x_1, \dots, x_n \rangle$  una sequenza.

Una sottosequenza  $Z = \langle z_1, \dots, z_k \rangle$  con  $k \leq n$  è una sequenza costituita dagli indici presi in ordine strettamente crescente dei caratteri di  $X$ .

*Esempio:*

$X = \langle A, C, D, B, A \rangle$

$Z_0 = \langle A, B \rangle = \langle 1, 4 \rangle$  è una sottosequenza di  $X$

$Z_1 = \langle A, B, B, A \rangle = \langle 1, 4, 4, 5 \rangle$  non è una sottosequenza di  $X$ , in quanto  $X$  non contiene due  $B$

$Z_2 = \langle A, B, C, D \rangle = \langle 1, 4, 2, 3 \rangle$  non è una sottosequenza di  $X$ , in quanto gli indici non sono presi in ordine strettamente crescente.

Consideriamo ora due sequenze  $X = \langle x_1, \dots, x_n \rangle$  e  $Y = \langle y_1, \dots, y_m \rangle$ ;

diremo che  $Z$  è **sottosequenza comune** se è contemporaneamente sottosequenza di  $X$  e di  $Y$ .

*Esempio:*

$X = \langle A, C, D, B \rangle$      $Y = \langle D, C, B, A \rangle$

$Z_0 = \langle C, B \rangle$  è sottosequenza comune a  $X$  e  $Y$ .

## LCS

Il problema della LCS vuole trovare una sottosequenza comune a due sequenze date che sia la più lunga possibile (*dunque anche questo è un problema di ottimizzazione*).

Proviamo a fare un ragionamento di tipo ricorsivo

Confrontiamo, nel caso generico,  $x_i$  con  $y_j$ , caratteri  $i$ -esimo e  $j$ -esimo delle sequenze  $X$  e  $Y$

- Se  $x_i = y_j$      $LCS(\langle x_1, \dots, x_{i-1} \rangle, \langle y_1, \dots, y_{j-1} \rangle) + 1$   
se i caratteri sono uguali è ovvio che mi conviene considerarli come facenti parte di  $Z$
- Se  $x_i \neq y_j$      $\max\{LCS(\langle x_1, \dots, x_{i-1} \rangle, \langle y_1, \dots, y_j \rangle), LCS(\langle x_1, \dots, x_i \rangle, \langle y_1, \dots, y_{j-1} \rangle)\}$   
se sono diversi, cerco di capire quale dei due caratteri mi conviene scartare

Definiamo quindi il problema:

**ISTANZA:**  $X = \langle x_1, \dots, x_n \rangle, Y = \langle y_1, \dots, y_m \rangle$

**SOLUZIONE:** La sottosequenza comune ad  $X$  e  $Y$  che sia la più lunga possibile.

## SOTTOPROBLEMI

Cerchiamo di suddividere il problema in sottoproblemi di taglia minore, definendo quindi il generico sottoproblema  $(i, j)$  come tale: la lunghezza della più lunga sottosequenza comune considerando solo i primi  $i$  caratteri di  $X$  ed i primi  $j$  caratteri di  $Y$ .

Avremo quindi  $(m + 1)(n + 1)$  sottoproblemi.

## DEFINIZIONE VARIABILI

Definiamo  $M[i][j]$  come la soluzione al sottoproblema di taglia  $(i, j)$ , ovvero come la lunghezza massima della LCS che riesco ad ottenere considerando solo i primi  $i$  caratteri di  $X$  ed i primi  $j$  caratteri di  $Y$ .

Abbiamo detto di avere  $(n + 1)(m + 1)$  sottoproblemi, di conseguenza la dimensione della nostra matrice sarà  $M[n + 1][m + 1]$ .

## CASO BASE

Se  $i = 0 \vee j = 0$

$$LCS(x_i, y_j) = 0$$

Ho in totale  $(m + n + 1)$  casi base.

## CASO PASSO

Se  $i > 0 \wedge j > 0$

$$LCS(x_i, y_j) = \begin{cases} LCS(x_{i-1}, y_{j-1}) + 1 & \text{se } x_i = y_j \\ \max\{LCS(x_{i-1}, y_j), LCS(x_i, y_{j-1})\} & \text{se } x_i \neq y_j \end{cases}$$

**SOLUZIONE:** La soluzione (*parziale*) la troveremo nella cella  $M[n][m]$

La soluzione è parziale nel caso ci interessi sapere anche quale sia la LCS oltre alla sua lunghezza, in tal caso possiamo salvarci in un'altra matrice  $(n + 1)(m + 1)$  gli spostamenti fatti nella prima matrice (se il risultato  $(i, j)$  proviene dalla diagonale o da riga/colonna) per poi usarla per sapere quali caratteri abbiamo usato, o ripercorrere la matrice originale per trovare i simboli (operazione più costosa in termini computazionali).

## ALGORITMO

$LCS(X, Y)$  {

  CASO BASE

    for  $i = 0$  to  $n$

$M[i][0] = 0$

    for  $j = 0$  to  $m$

$M[0][j] = 0$

  CASO PASSO

    for  $i = 1$  to  $n$

      for  $j = 1$  to  $m$

        if  $x[i] == y[j]$

$M[i][j] = M[i - 1][j - 1] + 1$

        else

$M[i][j] = \max(M[i][j - 1], M[i - 1][j])$

    return  $M[n][m]$

}

## TEMPO E SPAZIO

Il tempo di calcolo  $T(n, m)$  sarà

$$T(n, m) = (n + m) + (n * m) = \Theta(n * m)$$

Se includessimo anche la risalita per trovare la sequenza il tempo asintomatico non cambierebbe ( $\dots + (n + m)$ ).

Lo spazio  $S(n, m)$  sarà

$$S(n, m) = (n * m) = \Theta(n * m)$$

Se includessimo anche la seconda matrice lo spazio asintotico non cambierebbe ( $\dots + (n * m)$ ).

Notiamo però come l'algoritmo usi sempre solo due righe della colonna per eseguire i suoi calcoli, se ci interessasse solo la lunghezza della LCS potremmo ridurre lo spazio a solo due righe rendendolo lineare.

## LONGEST INCREASING SUBSEQUENCE

---

Abbiamo visto come identificare una sottosequenza comune a due sequenze che sia la più lunga possibile possa essere non immediato, ma concentriamoci ora su una sola sequenza: una sua sottosequenza è un qualunque suo carattere e la più lunga è, banalmente, la sequenza stessa. Ma non appena introduciamo ulteriori vincoli il problema può diventare complesso anche considerando una sola sequenza.

Il problema LIS consiste nel trovare una sottosequenza della sequenza  $X$  che sia la più lunga sottosequenza possibile prendendo solo **caratteri strettamente crescenti** (quindi ad esempio  $< A, C, B, D >$  non va bene perché  $C > B$ ).

Il classico approccio è provare tutte le possibili combinazioni, ma questo richiederebbe un tempo troppo elevato.

Definiamo quindi il problema come segue:

**ISTANZA:**  $X = \langle x_1, \dots, x_n \rangle$

**SOLUZIONE:** La più lunga sottosequenza di  $X$  strettamente crescente.

### SOTTOPROBLEMI

Cerchiamo quindi i sottoproblemi avendo in input una sequenza  $X = \langle x_1, \dots, x_n \rangle$ ;

Il sottoproblema di taglia  $i$  è: trovare la più lunga sottosequenza strettamente crescente che termini con l' $i$ -esimo carattere di  $X$ .

È importante notare come, a differenza del problema LCS, nel LIS, per dar risposta ai sottoproblemi, dobbiamo considerare tutti i risultati già calcolati; difatti se aggiungessi dei caratteri ad  $X$  potrei migliorare una LIS che magari prima non era ottimale e farla diventare la più lunga.

Ragionando allora potremo segnarci, per ogni carattere, quale sia la più lunga sottosequenza strettamente crescente che riusciamo a trovare che finisca con quello stesso carattere.

Avremo quindi che per il primo carattere il massimo è 1, in quanto il sottoproblema di taglia 1 considera solo un carattere, mentre, ipotizzando di aver già risolto i sottoproblemi più piccoli, ci basta vedere quale sia la massima lunghezza di una LIS compatibile con il carattere  $x_i$  (e quindi che sia minore di  $x_i$ ) e aggiungerci 1 (relativo al carattere  $x_i$ ).

### DEFINIZIONE VARIABILI

Definiamo  $L[i]$  come la soluzione al sottoproblema di taglia  $i$ , ovvero come la lunghezza della più lunga sottosequenza strettamente crescente che termini con l' $i$ -esimo carattere di  $X$ .

**Attenzione:** considero solo la sottosequenza più lunga che **termini** con il carattere  $i$ -esimo, perché devo controllare quale sia il miglior risultato che posso ottenere in ogni posizione, questo perché non so quali siano i prossimi caratteri e come possano influenzare il mio ottimo,

potenzialmente cambiandolo totalmente (*mentre nella LCS i risultati successivi potevano solo incrementare l'ottimo già trovato*).

## CASO BASE

$i = 1$

$L[1] = 1$  il massimo che posso fare con una sottosequenza di  $X$  che termini col suo primo carattere è certamente 1

Ho dunque un solo caso base.

## CASO PASSO

$1 < i \leq n$

$L[i] = 1 + \max\{ L[j] \mid 0 \leq j < i \mid x_j < x_i \}$

**SOLUZIONE:**  $\max\{ L[i] \mid 0 \leq i \leq n \}$

Questa volta la soluzione non la troveremo sempre nella stessa posizione come nella LCS, ma ci sarà data dal massimo tra le soluzioni ai sottoproblemi calcolati.

## ALGORITMO

LIS(X) {

  CASO BASE

$L[1] = 1$

  CASO PASSO

    for  $i = 2$  to  $n$

$max = 0$

      for  $j = 0$  to  $i$

        if  $L[j] > max$  AND  $X[j] < X[i]$

$max = L[j]$

$L[i] = max + 1$

  return max(L)

}

## TEMPO E SPAZIO

Il tempo dell'algoritmo è facilmente calcolabile ed è pari a  $T(n) = \Theta(n^2)$ .

Per evitare di scansionare nuovamente l'array per trovare il massimo totale, potremmo calcolarlo mentre costruiamo l'array controllando ad ogni fine ciclo interno se  $L[i]$  sia maggiore del massimo totale trovato fino ad allora, e nel caso aggiornarlo.

Lo spazio è dato dall'array  $L$  ed è pari a  $S(n) = \Theta(n)$ .

Per risolvere il problema originale (*fornire la sottosequenza*), potremmo ripercorrere l'array  $L$  all'indietro controllando, ogni volta che troviamo un valore minore di quello analizzato, se il carattere nella posizione di tale valore sia compatibile con quello nella posizione che stiamo analizzando.

Si potrebbe anche salvare in un altro array  $S$  di dimensione  $n$  quale carattere ha determinato il massimo nella posizione  $i$ , a seconda di cosa abbiamo bisogno di ottimizzare, spazio o tempo.

## GROWING LCS

---

Abbiamo fino ad ora visto i problemi della LCS e della LIS e ci chiediamo ora se sia possibile stabilire, tra due sequenze  $X = \langle x_1, \dots, x_n \rangle$  e  $Y = \langle y_1, \dots, y_m \rangle$  una *più lunga* sottosequenza comune che sia anche *crescente*, combinando quindi i due problemi precedenti.

Cerchiamo ora di definire il valore di un generico sottoproblema ipotizzando di aver già risolto tutti i sottoproblemi di taglia minore, ovvero i problemi di taglia  $(h, k)$  con:

$$0 \leq h < n \quad \text{e} \quad 0 \leq k < m$$

Consideriamo  $Z_{h,k}$  come la soluzione del problema  $(h, k)$ , se la conoscessi potrei decidere se aggiungere o meno il carattere, purtroppo però non so come sia fatta  $Z_{h,k}$  perché mi manca dell'informazione.

## PROBLEMA AUSILIARIO

Introduciamo quindi un problema ausiliario che ci permetta di ricostruire poi la soluzione del problema originale formulato come segue:

Date due sequenze  $X$  e  $Y$ , la soluzione è data dalla lunghezza della più lunga sottosequenza comune crescente che termini con  $x_n = y_m$ .

**ISTANZA:**  $X = \langle x_1, \dots, x_n \rangle$        $Y = \langle y_1, \dots, y_m \rangle$

**SOLUZIONE:** Lunghezza della più lunga sottosequenza comune e strettamente crescente di  $X$  e  $Y$  che *termini con  $x_n$  e  $y_m$  se questi coincidono*

## SOTTOPROBLEMI

Identifichiamo ora il sottoproblema generico del nostro problema ausiliario denotandolo con  $(i, j)$  definito come tale:

Date  $X$  e  $Y$ , determinare la lunghezza della più lunga sottosequenza crescente che termini coi caratteri  $x_i$  e  $y_j$  se questi coincidono.

Avremo quindi in totale  $(n + 1)(m + 1)$  sottoproblemi.

## DEFINIZIONE VARIABILI

Definiamo quindi  $C_{i,j}$  come la soluzione al sottoproblema di taglia  $(i, j)$ , ovvero come la lunghezza della più lunga sottosequenza comune ad  $X$  e  $Y$  che termini con i caratteri  $x_i$  e  $y_j$  se questi coincidono.

Avendo  $(n + 1)(m + 1)$  sottoproblemi,  $C$  sarà definita come una matrice di dimensione  $(n + 1)(m + 1)$ .

Ora, cerchiamo di calcolare  $C_{i,j}$  con  $i, j > 0$  assumendo di aver già risolto i sottoproblemi di taglia minore,

ovvero quelli identificati da  $(h, k)$  con  $0 \leq h < i$     e     $0 \leq k < j$ :

Avremo dunque che ogni qual volta le  $x_i$  e  $y_j$  considerati siano diverse, la risposta al sottoproblema è 0

*(poiché non esiste una sottosequenza comune che termini sia con  $x_i$  che con  $y_j$ )*

Mentre quando sono uguali, le aggiungo al meglio che sono riuscito ad ottenere i sottoproblemi precedenti.



## CASO BASE

$\forall (i, j) \text{ con } x_i \neq y_j, C_{i,j} = 0$

In questo caso il numero di casi base dunque dipende dall'input.

## CASO PASSO

$\forall (i, j) \text{ con } x_i = y_j$

$C_{i,j} = 1 + \max\{C_{h,k} \mid 0 \leq h < i \text{ e } 0 \leq k < j \mid x_h < x_i\} \text{ con } \max(\emptyset) = 0$

(potrei aggiungere 'AND  $y_k < y_j$ ' alla condizione, ma sarebbe ridondante in quanto  $x_i = y_j$ )

**SOLUZIONE PROBLEMA ORIGINALE:**  $\max\{C_{i,j}\}$  con  $1 \leq i \leq n$  e  $1 \leq j \leq m$

## ALGORITMO

```
C[N][M]{
  for i = 0 to n
    for j = 0 to m
      if X[i] != Y[j]
        C[i][j] = 0      CASO BASE
      else
        max = 0          CASO PASSO
        for h = 0 to i - 1
          for k = 0 to j - 1
            if X[h] < X[i] AND C[h][k] > max
              max = C[h][k]
```

$$C[i][j] = 1 + \max$$

```
  return Max(C)
```

```
}
```

## TEMPO E SPAZIO

Si può subito notare come l'algoritmo ci impieghi  $T(n, m) = \Theta(m^2 * n^2)$

Mentre lo spazio è pari a  $S(n, m) = \Theta(m * n)$

Potremmo ridurre un po' il tempo (anche se quello asintotico non cambierebbe) calcolando il massimo totale mentre costruiamo la matrice invece che riscansionarla alla fine per trovare il massimo.

**LCS LUNGHEZZA  $\leq L$**

## ISTANZA

$X = \langle x_1, x_2 \dots x_m \rangle$  m

$Y = \langle y_1, y_2 \dots y_n \rangle$  n

$L \in \mathbb{N}$

**SOLUZIONE:** true SSE  $\exists$  una LCS(X, Y) di lunghezza  $\geq L$

sotto problema: (i, j, l) con  $i \in \{0, 1, \dots, m\}$ ,  $j \in \{0, 1, \dots, n\}$ ,  $l \in \{0, 1, \dots, L\}$

istanza  $X_i = \langle x_1 \dots x_i \rangle$   $Y_j = \langle y_1 \dots y_j \rangle$  l

soluzione true SSE  $\exists$  LCS( $X_i, Y_j$ ) di lunghezza  $\geq l$

In totale ho  **$(m + 1)(n + 1)(l + 1)$**  sotto problemi

## DEFINIZIONE VARIABILI COINVOLTE

$\forall (i, j, l)$  con  $i \in \{0, 1, \dots, m\}$ ,  $j \in \{0, 1, \dots, n\}$ ,  $l \in \{0, 1, \dots, L\}$

**$C_{i,j,l}$**  è definita come la soluzione del sotto problema (i, j, l)  $\implies$  vale true SSE una LCS( $X_i, Y_j$ ) con lunghezza  $\geq l$

## CASO BASE

$$C_{i,j,l} = \begin{cases} True & \text{se } l = 0 \text{ qualunque sia il valore di } i \text{ e } j \\ False & \text{se } l > 0 \wedge (i < l \vee j < l) \end{cases}$$

**Quanti sono i casi base?**

- True: (i, j, 0)  $\rightarrow (m+1)(n+1)$
- False: (i, j, l)  $\rightarrow$  homework

## CASO RICORSIVO

(i, j, l)  $l > 0 \wedge (i \geq l \vee j \geq l)$

Dato (i, j, l) assumo di aver già risolto i sotto problemi più piccoli di (i, j, l)

$$C_{i,j,l} = \begin{cases} C_{i-1,j-1,l-1} & \text{se } x_i = y_j \\ C_{i-1,j,l-1} \vee C_{i,j-1,l-1} & \text{se } x_i \neq y_j \end{cases}$$

**Soluzione problema di partenza =  $C[m][n][l]$**

## ALGORITMO

Algo(X, Y, L) {

*Caso base*

  for i = 0 to m

    for j = 0 to n

$C[i][j][l] = \text{True}$

```

Caso ricorsivo
for l = 1 to L
  for i = 1 to m
    for j = 0 to n
      if (i < l) ∨ (j < l)
        C[i][j][l] = False
      else
        if xi = yj
          C[i][j][l] = C[i-1][j-1][l-1]
        else
          C[i][j][l] = (C[i-1][j][l]) ∨ (C[i][j-1][l])
    Return C[M][N][L];
}

```

## LUNGHEZZA LCS CON COLORI

---

### ISTANZA

$X = \langle x_1, x_2, \dots, x_m \rangle$  m

$Y = \langle y_1, y_2, \dots, y_n \rangle$  n

$K = 3$

Ad ogni carattere di  $\Sigma$  è associato il colore rosso o blu mediante  $col: \Sigma \rightarrow \{R, B\}$

**SOLUZIONE:** Lunghezza di LCS nella quale ci sono al massimo K caratteri a cui è associato il colore rosso

Sotto problema (i, j, k) con  $i \in \{0, 1, \dots, m\}$ ,  $j \in \{0, 1, \dots, n\}$ ,  $k \in \{0, 1, \dots, K\}$

istanza  $X_i = \langle x_1 \dots x_i \rangle$   $Y_j = \langle y_1 \dots y_j \rangle$  k

soluzione: lunghezza di una LCS( $x_i, y_j$ ) nella quale vi sono al massimo k caratteri rossi

### DEFINIZIONE VARIBILI COINVOLTE

$\forall (i, j, l)$  con  $i \in \{0, 1, \dots, m\}$ ,  $j \in \{0, 1, \dots, n\}$ ,  $l \in \{0, 1, \dots, L\}$

$C_{i,j,k}$  è definita come la soluzione del sotto problema (i, j, k)  $\implies$  Lunghezza LCS con al massimo k rossi

## CASO BASE

$C_{i,j,k} = 0$  per  $i = 0 \vee j = 0 \forall k \in \{0, \dots, K\}$

## CASO RICORSIVO

$(i, j, k) \quad i > 0 \wedge j > 0 \wedge \forall k \in \{0, \dots, K\}$

$$C_{i,j,k} = \begin{cases} 1 + C_{i-1, j-1, k-1} & \text{Se } x_i = y_j \wedge Col(x_i) = Red \wedge k > 0 \\ C_{i-1, j-1, k} & \text{Se } x_i = y_j \wedge Col(x_i) = Red \wedge k = 0 \\ 1 + C_{i-1, j-1, k} & \text{Se } x_i = y_j \wedge Col(x_i) = Blue \\ \max(C_{i-1, j, k}, C_{i, j-1, k}) & \text{Se } x_i \neq y_j \end{cases}$$

**Soluzione problema di partenza =  $C[M][N][K]$**

Da fare: invece che "al massimo"

- **almeno**  $K$  rossi

## ESATTAMENTE K

### CASO BASE

$C_{i,j,k} = 0$  per  $(i = 0 \vee j = 0) \wedge k = 0$

$C_{i,j,k} = -\infty$  per  $(i = 0 \vee j = 0) \wedge k > 0$  (Non c'è nessuna LCS con  $k$  rossi, nemmeno quella vuota)

Dove  $-\infty$  è un simbolo tale che qualsiasi cosa sommata ad esso dia sempre  $-\infty$

Tutto sto casino perché se prendo ad esempio  $X = \langle A \rangle$  e  $Y = \langle A \rangle$  con  $col(A) = Blue$  e  $K > 0$ , l'algoritmo di prima darebbe 1, mentre in questo caso la LCS con  $K$  rossi non esiste

### CASO RICORSIVO

$(i, j, k) \quad i > 0 \wedge j > 0 \wedge \forall k \in \{0, \dots, K\}$

$$C_{i,j,k} = \begin{cases} 1 + C_{i-1, j-1, k-1} & \text{Se } x_i = y_j \wedge Col(x_i) = Red \wedge k > 0 \\ C_{i-1, j-1, k} & \text{Se } x_i = y_j \wedge Col(x_i) = Red \wedge k = 0 \\ 1 + C_{i-1, j-1, k} & \text{Se } x_i = y_j \wedge Col(x_i) = Blue \\ \max(C_{i-1, j, k}, C_{i, j-1, k}) & \text{Se } x_i \neq y_j \end{cases}$$

# SEQUENZA INTERLEAVING

## ISTANZA

$$X = \langle x_1, x_2 \dots x_m \rangle \text{ m}$$

$$Y = \langle y_1, y_2 \dots y_n \rangle \text{ n}$$

$$W = \langle w_1, w_2 \dots w_{m+n} \rangle$$

**SOLUZIONE:** True SSE W è interleaving di X e Y ossia se X e Y sono due sottosequenze disgiunte di W

*Esempio:*  $X = \langle C, I, A, O \rangle$   $Y = \langle M, A, M, M, A \rangle$   $W = \langle C, I, M, A, A, M, M, A, O \rangle$

*Devono però essere disgiunte, una lettera non può essere parte di entrambe le sottosequenze*

Sotto problema

$$\text{istanza } X_i = \langle x_1 \dots x_i \rangle \quad Y_j = \langle y_1 \dots y_j \rangle \quad W_{i+j} = \langle w_1 \dots w_{i+j} \rangle$$

*Soluzione:* true sse  $W_{i+j}$  è interleaving di  $X_i$  e  $Y_j$

## DEFINIZIONE VARIBILI COINVOLTE

$\forall (i, j)$  con  $i \in \{0, 1, \dots, m\}$ ,  $j \in \{0, 1, \dots, n\}$

$C_{i,j}$  è definita come la soluzione del sotto problema  $(i, j)$  true sse  $W_{i+j}$  è interleaving di  $X_i$  e  $Y_j$

## CASO BASE

**Se  $i = 0 \wedge j = 0$**

$C_{i,j} = \text{True}$  (e quindi  $W_0 = \epsilon$ )

**Se  $i = 0 \wedge j > 0$**

$C_{i,j} = \text{False}$  se  $w_j \neq y_j$  (ovvero se  $i = 0$ , W deve essere uguale a Y per far sì che sia interleaving)

**Se  $i > 0 \wedge j = 0$**

$C_{i,j} = \text{False}$  se  $w_i \neq x_i$  (ovvero se  $j = 0$ , W deve essere uguale a X per far sì che sia interleaving)

**Se  $i > 0 \wedge j > 0$**

$C_{i,j} = \text{False}$  se  $w_{i+j} \neq x_i \wedge w_{i+j} \neq y_j$

## CASO RICORSIVO

**Se  $i = 0 \wedge j > 0$**

$C_{i,j} = C_{i,j-1}$  se  $w_j = y_j$

**Se  $i > 0 \wedge j = 0$**

$C_{i,j} = C_{i-1,j}$  se  $w_i = x_i$

**Se  $i > 0 \wedge j > 0$**

$C_{i,j} = C_{i-1,j}$  se  $w_{i+j} = x_i \wedge w_{i+j} \neq y_j$

$C_{i,j} = C_{i,j-1}$  se  $w_{i+j} \neq x_i \wedge w_{i+j} = y_j$

$C_{i,j} = C_{i-1,j} \vee C_{i,j-1}$  se  $w_{i+j} = x_i \wedge w_{i+j} = y_j$

# MIX TRA LCS E KNAPSACK

---

## ISTANZA

$X = \langle x_1, x_2, \dots, x_m \rangle$

$Y = \langle y_1, y_2, \dots, y_n \rangle$

Sequenze di numeri  $\mathbb{N}$

**SOLUZIONE:** Lunghezza di una LCS tra X e Y tale che la somma dei numeri è  $\leq K$  (dove  $K \geq 0$ )

## DEFINIZIONE VARIBILI COINVOLTE

**ISTANZA:**  $(i, j, k)$  con  $i \in \{0, 1, \dots, m\}$ ,  $j \in \{0, 1, \dots, n\}$ ,  $k \in \{0, 1, \dots, K\}$

$\forall (i, j, k)$  con  $i \in \{0, 1, \dots, m\}$ ,  $j \in \{0, 1, \dots, n\}$ ,  $k \in \{0, 1, \dots, K\}$

Introduco la variabile  $C_{i,j,k}$  che è soluzione del sotto problema  $(i, j, k)$  ossia è la lunghezza di una LCS tra  $x_i, y_j$  t.c. la somma dei suoi valori sia  $\leq k$

## CASO BASE

$C_{i,j,k} = 0$  per  $(i = 0 \vee j = 0) \quad \forall k \in \{0, \dots, K\}$

## CASO RICORSIVO

$$C_{i,j,k} = \begin{cases} C_{i-1,j-1,k} & \text{Se } x_i = y_j \wedge x_i > k \\ 1 + C_{i-1,j-1,k-i} & \text{Se } x_i = y_j \wedge x_i \leq k \\ \max(C_{i-1,j,k}, C_{i,j-1,k}) & \text{Se } x_i \neq y_j \end{cases}$$

(Somiglia molto al problema dello zaino e/o alla subset sum)

# LICS

---

## ISTANZA

$X = \langle x_1, x_2 \dots, x_m \rangle$  m

$Y = \langle y_1, y_2 \dots, y_n \rangle$  n

**SOLUZIONE:** lunghezza di LCS tra X e Y *crescente*

## DEFINIZIONE VARIBILI COINVOLTE

### MANCA INFORMAZIONE PER I SOTTO PROBLEMI

Si introduce un problema ausiliario (AUX)

**ISTANZA:**  $x_m, y_n$

**SOLUZIONE:** Lunghezza di LCS di X e Y crescente e che termina con  $x_m$  e  $y_n$  (se questi coincidono)

$C_{i,j}$  soluzione del sotto problema (i, j) del problema aux  $\implies$  lunghezza di una LCS di X e Y e che termina con  $x_i$  e  $y_j$  se coincidono

## CASO BASE (AUX)

$C_{i,j} = 0$  se  $x_i \neq y_j$

## CASO RICORSIVO (AUX)

$C_{i,j} = 1 + \max\{C_{h,k} \mid 1 \leq h \leq i, 1 \leq k \leq j \mid x_h < x_i\}$  Se  $x_i = y_j$  (uguale per ordine lessicografico)

(per decrescente basta mettere  $x_h > x_i$ , uguale per ordine lessicografico decrescente)

**SOLUZIONE PROBLEMA DI PARTENZA:**  $\max\{C_{i,j} \mid i \in \{0, 1, \dots, m\}, j \in \{0, 1, \dots, n\}\}$

# VARIANTI

---

## CARATTERI UGUALI

### ISTANZA

$X = \langle x_1, x_2 \dots, x_m \rangle$  m

$Y = \langle y_1, y_2 \dots, y_n \rangle$  n

**SOLUZIONE:** lunghezza di LCS tra X e Y senza 2 caratteri consecutivi uguali

**TUTTO UGUALE MA:**

### CASO RICORSIVO (AUX)

$C_{i,j} = 1 + \max\{C_{h,k} \mid 1 \leq h \leq i, 1 \leq k \leq j \mid x_h \neq x_i\}$  Se  $x_i = y_j$

## PARI E DISPARI

### ISTANZA

$X = \langle x_1, x_2, \dots, x_m \rangle$  m

$Y = \langle y_1, y_2, \dots, y_n \rangle$  n

**SOLUZIONE:** lunghezza di LCS tra X e Y che alterna pari e dispari

**TUTTO UGUALE MA:**

### CASO RICORSIVO (AUX)

$C_{i,j} = 1 + \max\{C_{h,k} \mid 1 \leq h \leq i, 1 \leq k \leq j \mid x_h \% 2 \neq x_i \% 2\}$  Se  $x_i = y_j$

## ALTERNA BLU E ROSSO

*(Il Denny ci tiene a specificare che tifa Genova, non ci capisco un cazzo di calcio io però)*

### ISTANZA

$X = \langle x_1, x_2, \dots, x_m \rangle$  m

$Y = \langle y_1, y_2, \dots, y_n \rangle$  n

**SOLUZIONE:** lunghezza di LCS tra X e Y che alterna blu e rossi

**TUTTO UGUALE MA:**

### CASO RICORSIVO (AUX)

$C_{i,j} = 1 + \max\{C_{h,k} \mid 1 \leq h \leq i, 1 \leq k \leq j \mid \text{Col}(x_h) \neq \text{Col}(x_i)\}$  Se  $x_i = y_j$

*("Per me esiste solo un pesto, ovvero il genovese, oggi lo mangio che l'ho fatto io" cit. Denny)*

*(Also, il riso java lo mangia quando ha il cagotto e odia il pesto dell'esselunga)*

## A CONSECUTIVE

### ISTANZA

$X = \langle x_1, x_2, \dots, x_m \rangle$  m

$Y = \langle y_1, y_2, \dots, y_n \rangle$  n

Caratteri

**SOLUZIONE:** lunghezza di LCS tra X e Y nella quale non vi sono mai lettere 'A' consecutive

**TUTTO UGUALE MA:**

### CASO RICORSIVO (AUX)

$C_{i,j} = \begin{cases}$

$1 + \max\{C_{h,k} \mid 1 \leq h \leq i, 1 \leq k \leq j \mid x_h \neq x_i\}$  Se  $x_i = y_j$  e  $x_i \neq 'A'$

$1 + \max\{C_{h,k} \mid 1 \leq h \leq i, 1 \leq k \leq j \mid x_h \neq x_i\}$  Se  $x_i = y_j$  e  $x_i = 'A'$

$\end{cases}$



